Porte de um Simulador da Eletrofisiologia Cardíaca para um Ambiente de Grade Computacional

Micael Peters Xavier

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Orientador: Marcelo Lobosco
Co-orientador: Rodrigo Weber dos Santos

Juiz de Fora, MG
Dezembro de 2010

Micael Peters Xavier

Porte de um Simulador da Eletrofisiologia Cardíaca para um Ambiente de Grade Computacional

Orientador:
Marcelo Lobosco

Co-orientador:
Rodrigo Weber dos Santos

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Juiz de Fora, MG

Dezembro de 2010

Porte de um Simulador da Eletrofisiologia Cardíaca para um Ambiente de Grade Computacional

Micael Peters Xavier

Monografia submetida ao corpo docente do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como parte integrante dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação

---

Prof. Marcelo Lobosco, D. Sc.
Orientador

Prof. Rodrigo Weber dos Santos, D. Sc.
Co-orientador

Prof. Rafael Sachetto Oliveira, M. Sc.

AGRADECIMENTOS

Ter a quem agradecer é uma graça divina, pois isto significa que ao longo dessa caminhada não estive sozinho e que comigo estiveram todos aqueles que de alguma forma desejaram esta vitória.

Meus sinceros agradecimentos a minha família, principalmente minha mãe Cláudia, pai Antônio, irmão Maicom e avó Lourdes por terem me dado todo o amor e educação durante minha vida, sempre me ajudando e apoiando em minhas decisões.

A minha noiva Laís, por seu amor, carinho e apoio dados a mim em todos os momentos do nosso relacionamento juntos.

Agradeço também aos meus professores Marcelo Lobosco e Rodrigo Weber dos Santos pela amizade, ensinamentos e orientação dados durante esses últimos anos.

Aos meus amigos de curso e do laboratório FISIOCOMP: Ricardo, Gustavo, Abraão, Alexandre, Bernardo, Victor, Daniel, Rafael, Raphael, Muilaerte, Tárcio, Nilton, Wellington, Paulo, Márcio, Felipe, Humberto, Henrique, Roberto, Carlos, entre outros, pela amizade, experiência e momentos de descontração vividos.

Aos professores do DCC e a todos os meus colegas da UFJF, por todo o incentivo prestado.

Resumo

Modelos matemático-computacionais para a representação do fenômeno de propagação de eletricidade pelo coração são largamente empregados com o objetivo de estudar o comportamento deste órgão quando exposto a substâncias químicas, permitindo, assim, o teste *in silico* do efeito de novos medicamentos. Entretanto, o custo computacional para a execução destes modelos é elevado. Uma alternativa para reduzir o tempo de execução é o emprego de técnicas de paralelismo. Neste trabalho, apresentamos a paralelização, em um ambiente de grade computacional, de uma aplicação que simula o fenômeno de propagação de eletricidade no coração. Grade computacional é uma infraestrutura composta por recursos computacionais que disponibiliza diversos serviços, como por exemplo, poder computacional para execução de aplicações remotamente. Utilizando esses recursos, resultados preliminares puderam ser avaliados. Estes mostram que a tecnologia de grades computacionais é promissora e permite reduzir o tempo total de execução da aplicação em questão.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A compreensão de fenômenos complexos é de vital importância para os mais distintos campos do conhecimento humano. Muitas vezes tais fenômenos correspondem a uma multiplicidade de sistemas entrelaçados, que, continuamente, interagem e evoluem, o que exige um olhar multidisciplinar para o seu completo entendimento. A modelagem computacional é, muitas vezes, empregada com o objetivo de analisar, representar e simular tais fenômenos com o auxílio de computadores.

Após ser observado, um fenômeno pode ser representado através de equações matemáticas, que, por sua vez, podem ser solucionadas por um computador. Este conjunto de equações é chamado de modelo. Por via de regra, quanto mais próximo de um fenômeno real, mais complexo o modelo, fazendo com que este se torne mais difícil e custoso de ser resolvido computacionalmente. Este custo pode ser reduzido com o emprego de técnicas de computação paralela.

Segundo dados do Ministério da Saúde [35], as maiores causas de morte no Brasil são as doenças do aparelho circulatório, com 27,88% do total no ano de 2004. Estas doenças compreendem infartos agudos do miocárdio, doenças hipertensivas, entre outros problemas no coração e no aparelho circulatório. Pesquisadores de diversas áreas vêm estudando sobre o assunto, buscando entender as causas e descobrir novas curas para essas doenças.

Neste trabalho é apresentado o porte para um ambiente de grade computacional de uma aplicação que permite a simulação da propagação de correntes elétricas pelo músculo cardíaco. Grades computacionais são infraestruturas de *hardware* e *software* que fornecem acesso confiável, consistente, universal e barato a recursos compartilhados que estão localizados em diversas instituições ao redor do globo. Os recursos disponibilizados são basicamente poder de processamento e armazenamento de arquivos em grande escala. O ambiente de grade utilizado é o EELA-2 e o *middleware* que o gerencia é o gLite. Este *middleware* é responsável por manter a interoperabilidade entre arquiteturas distribuídas

e a comunicação em um meio heterogêneo, conectando aplicações e mantendo o serviço confiável e disponível. Este *software* permite submeter tarefas (*jobs*) para execução nos computadores remotos (ou nós) que compõem esta grade. Resultados preliminares indicam que o uso de grades computacionais para execução de aplicações que demandam alto desempenho, podem ajudar de modo significativo, reduzindo o tempo de execução dessas aplicações.

O trabalho está organizado da seguinte maneira: o capítulo 2 descreve as características do ambiente de grade utilizado para a realização dos experimentos. O capítulo 3 apresenta o órgão coração, aborda conceitos da eletrofisiologia cardíaca e modelos celulares utilizados neste trabalho. No capítulo 4 são apresentados os principais conceitos necessários para o entendimento do trabalho, descrevendo os modelos celulares, algoritmos genéticos e a modelagem do problema inverso. No capítulo 4 também é descrita a metodologia utilizada para a criação dos arquivos que permitiram que a aplicação fosse executada na grade. Os resultados obtidos com os experimentos realizados são encontrados no capítulo 5. Por fim, o capítulo 6 conclui o trabalho apontando trabalhos futuros.

2 Computação em grade

2.1 Introdução

Computação em Grade é um serviço que possui aglomerados de recursos computacionais espalhados por todo o globo que vem ajudando pesquisadores de diversas áreas a executarem suas aplicações e armazenarem grande quantidade de dados remotamente. Os recursos oferecidos são acessados através da Internet por usuários previamente autorizados a utilizarem o serviço. Este serviço diferencia-se da *World Wide Web*, pois permite compartilhar recursos computacionais pela Internet, enquanto a *Web* apenas compartilha informações [26].

Neste capítulo aborda-se o conceito de computação em grade e o software responsável pelo seu gerenciamento, denominado *middleware*.

2.2 Grade computacional

Grades computacionais são infraestruturas de hardware e software que fornecem acesso confiável, consistente, universal e barato a recursos compartilhados [15]. Estes recursos estão localizados em instituições espalhadas pelo globo, e são constituídos de diversos elementos, desde processadores, passando por unidades de armazenamento de dados, até sensores e telescópios. O acesso a tais recursos é oferecido a usuários previamente autorizados a utilizar a grade. Os usuários estão dispostos em grupos chamados de Organizações Virtuais (VO), que definem diferentes permissões de acessos a determinados recursos.

Para que a grade ofereça seus serviços aos usuários, é necessário um conjunto de ferramentas de controle e gerenciamento dos recursos: o *middleware*, uma camada de software responsável pelo gerenciamento e segurança da grade.

O nome *grid* é uma analogia às redes elétricas (*electric power grid*). A idéia de utilização dos recursos em um *grid* é similar à de um usuário que necessite de energia

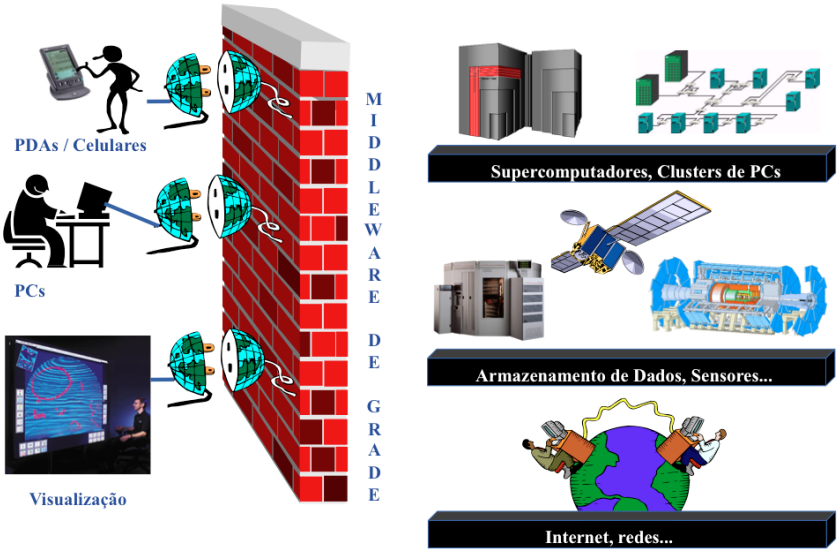

Figura 1: *Grid* computacional

elétrica para fazer o uso de um certo dispositivo. Ele o conecta a uma tomada e obtém energia sem saber de onde veio ou como esta foi produzida. Através de um PDA, por exemplo, conectado à internet, o usuário do *grid* pode executar aplicações e armazenar dados em computadores remotos de forma transparente, como ilustrado na Figura 1. Cabe ao *middleware* fornecer essa transparência ao usuário e gerenciar todo o ambiente.

O ambiente de grade utilizado é o EELA-2 (*E-science grid facility for Europe and Latin America*) [10]. EELA-2 é uma infraestrutura que consiste de quarenta e um centros de recursos (treze na Europa e vinte e oito na América Latina), com cerca de três mil núcleos de processamento (*cores*) de computação e mais de setecentos *Terabytes* de armazenamento em disco. O *middleware* responsável pelo controle desta grade é o gLite [16].

2.3 A camada *middleware*

O *middleware* é uma camada de *software* que tem por objetivo fazer a mediação entre o sistema operacional e demais aplicações, sendo o responsável por prover ferramentas para que os vários recursos participem da grade. De um modo geral, ele é utilizado para mover ou transportar informações e dados entre programas de diferentes protocolos de comunicação, plataformas e dependências do sistema operacional.

Os *middlewares* interconectam sistemas que são heterogêneos e possuem dispositivos distintos. Para fins de exemplificação, os sistemas mais comuns são sistemas de telecomu-

nicação e redes de computadores, onde neste meio pode-se destacar a Internet. Cabe ao *middleware* garantir a interoperabilidade entre as arquiteturas distribuídas, mantendo o serviço confiável e disponível. A localização de serviços e aplicações de forma transparente na rede também é um dos serviços oferecidos pelo *middleware*, camada esta que faz tudo parecer "simples de ser acessado" ao usuário final.

São exemplos de *middlewares*:

- *gLite* [16]
- *DIRAC* [8]
- *OurGrid* [36]
- *Globus* [25]

O *middleware* utilizado neste trabalho, que faz o gerenciamento da grade do EELA-2 é o gLite [16]. Mais informações sobre suas funcionalidades são descritas a seguir.

2.4 gLite

O projeto gLite é uma solução completa em grades computacionais para uma comunidades de usuários. gLite cria uma camada entre uma comunidade em particular e vários recursos computacionais, permitindo seu uso otimizado, transparente e confiável. Este *middleware* possibilita a execução da aplicação em máquinas remotas na grade, ambiente de alta heterogeneidade de dispositivos. É também de sua responsabilidade todo o processo de autenticação do usuário na grade, gerenciamento do proxy, armazenamento de arquivos e gerência de carga de trabalho.

2.4.1 Autenticação

Fornece uma camada de conexão segura entre os clientes e serviços no sistema. Baseado nos certificados X.509 e nas autoridades certificadoras, permite dar acesso aos usuários baseado em seus respectivos direitos de acesso aos recursos disponibilizados pela grade. Os certificados X.509 possuem uma chave pública e uma privada, onde a chave pública é conhecida por todos e a chave privada é de exclusividade de cada usuário. No processo de autenticação, o usuário que deseja fazer uso de um servidor recebe o certificado do servidor, que é a chave pública. Após receber a chave pública, o usuário faz a comparação

desta com a assinatura da autoridade certificadora, enviando depois um *token* aleatório ao servidor. Este criptografa o *token* recebido e envia o resultado como uma mensagem ao usuário. Por último, o usuário decodifica a mensagem recebida com a chave pública e a compara com seu *token* inicial. Se forem iguais, o usuário está apto a ser autenticado pelo serviço. O serviço de autenticação também oferece suporte aos mecanismos necessários para a criação de *proxy* e delegação.

2.4.2 Armazenamento de arquivos

Um dos principais serviços disponibilizados pelos *middlewares* é o gerenciamento de arquivos. Em um ambiente de grade podem existir diversas réplicas de arquivos espalhados por diferentes locais. De acordo com [17], o ideal seria que o usuário não precisasse saber onde seus arquivos estão armazenados, por isso o serviço de armazenamento de arquivos atribui um nome lógico ao arquivo (*Logical File Name* (LFN)), possibilitando o acesso ao dado a partir deste nome. *GRID Unique ID* (GUID) é um identificador único para um arquivo. O responsável por informar o local físico dos arquivos é o *Storage URL* (SURL) e *Transport URL* (TURL) fornece informações como o servidor, local, protocolo e porta para acessar os arquivos. O serviço de *LCG File Catalogue* (LFC) é que faz o mapeamento entre esses diferentes nomes, enquanto que os arquivos são armazenados nos *Storage Elements* (SEs). O SE é um serviço que permite que usuários ou aplicações armazenem arquivos no banco de dados.

Para manipulação de arquivos e diretórios na grade, vide apêndice A.2.

2.4.3 Gerenciamento do *proxy*

gLite provê um conjunto de ferramentas para a gerência de *proxies* [20]. Os *proxies* são certificados temporários assinados digitalmente pelo usuário que também possuem chaves pública e privada. O intuito em utilizá-los é permitir o acesso a dois recursos importantes da grade:

- *Single sign on* - mecanismo que permite ao usuário autenticar-se na grade;
- *Delegation* - diversos serviços da grade podem operar em nome do usuário sem que este faça interação alguma.

Além disso, existe a presença de um componente chamado VOMS - *Virtual Organization Membership Services*, que foi criado a partir da necessidade em dividir os usuários da

grade em grupos chamados de *Virtual Organizations* - VO [14]. Isso porque é necessário que existam diversos privilégios de acesso dos usuários aos recursos computacionais da grade. Desta forma este serviço faz o devido gerenciamento desses recursos para cada cliente de uma forma independente.

2.4.4 Gerência de carga de trabalho

Este serviço é responsável por enviar as tarefas à grade e permitir que seus resultados sejam entregues ao cliente. Para submeter uma tarefa (ou *job*) à grade, deve-se primeiro criar uma delegação, já que em todas as vezes que o serviço WMProxy [21] precisar interagir com outros serviços relacionados ao *job*, não seja necessário requisitar interações do usuário, utilizando para isso a delegação. Para criar uma delegação, pode-se utilizar o comando:

- glite-wms-job-delegate-proxy -d <delegID>,

onde <delegID> é uma *string* qualquer definida pelo usuário.

O próximo passo é a criação de um arquivo de descrição do trabalho (JDL - *Job Description Language*). Neste arquivo, por meio de *tags*, os requisitos necessários para que o trabalho seja executado são especificados. Abaixo são descritas algumas *tags* importantes:

- *JobType* - especifica qual é o tipo da tarefa;
- *Executable* - informa qual será o arquivo a ser executado, sendo geralmente um *script* ou a própria aplicação;
- *CPUNumber* - define o número de nós a ser utilizado;
- *InputSandBox* - todos os arquivos de entrada que a aplicação necessita para sua execução;
- *OutputSandBox* - todos os arquivos de saída que a aplicação gera.

Um exemplo de arquivo JDL pode ser visualizado na Figura 2. A *tag Arguments* fornece argumentos para o rótulo *Executable*; *StdOutput* define o arquivo de saída padrão da aplicação; *StdError* informa qual é o arquivo de erro padrão e *ShallowRetryCount* especifica o número de máximo de tentativas de execução da tarefa caso ocorra algum erro ou falha. Para maiores informações sobre a linguagem JDL e suas *tags*, vide [22].

```
[
    JobType             = "Normal";
    CPUNumber           = 5;
    Executable          = "bin/helloWorld";
    Arguments           = "ola";
    StdOutput           = "helloWorld.out";
    StdError            = "helloWorld.err";
    InputSandbox        = {"helloWorld"};
    OutputSandbox       = {"helloWorld.err" , "helloWorld.out"};
    ShallowRetryCount = 3;
]
```

Figura 2: Arquivo de descrição da tarefa - JDL

Uma vez que o cliente tenha feito a submissão de um *job* ao gLite WMS (*Workload Management System*), o serviço atribui um identificador de tarefa ($jobID$) que define unicamente o *job* do usuário na base de dados [24]. Esse $jobID$ pode ser utilizado para futuras operações no *job*, como por exemplo, a consulta de seu *status*, seu cancelamento e, também na obtenção dos arquivos de saída que a aplicação gera em caso de sucesso na execução. Estes arquivos são descritos na *tag* $OutputSandBox$.

3 Eletrofisiologia cardíaca

3.1 O coração

O coração humano é um órgão musculoso do tamanho de um punho e se responsabiliza pelo bombeamento do sangue através de todo o organismo, processo este que é feito em aproximadamente quarenta e cinco segundos quando o organismo está em repouso. O coração bate cerca de cem mil vezes por dia, bombeando aproximadamente sete mil e quinhentos litros de sangue [37].

Neste tempo, o órgão bombeia sangue suficiente a uma pressão razoável, para percorrer todo o corpo nos sentidos de ida e volta, transportando assim, oxigênio e nutrientes necessários às células que sustentam as atividades orgânicas. É constituído por duas porções: a metade direita ou coração direito, onde circula o sangue venoso e a metade esquerda, onde circula sangue arterial. Cada uma destas metades do coração é constituída por duas cavidades, uma superior, a aurícula, e uma inferior, o ventrículo. Estas cavidades comunicam entre si pelos orifícios auriculo-ventriculares. A circulação sanguínea é assegurada pelo batimento cardíaco, ou seja, pelo batimento do coração, que lança o sangue nas artérias.

A corrente sanguínea chega rica em gás carbônico no átrio direito, que funciona como uma bomba mais fraca. O sangue é impulsionado para o ventrículo direito, que então leva a circulação aos pulmões. Ali o gás carbônico é expelido e o sangue recebe oxigênio. A corrente passa pelo átrio esquerdo e segue para o ventrículo esquerdo, uma bomba potente que impulsiona o sangue para os órgãos periféricos. Este esquema é ilustrado na Figura 3.

Segundo [4], o coração dispõe de mecanismos especiais que geram e transmitem impulsos elétricos pelo miocárdio, de modo a manter a sucessão de contrações rítmicas. Quando este sistema funciona corretamente, os átrios contraem um sexto de segundo a frente dos ventrículos, permitindo que estes se encham de sangue e logo após o impulsionem para os

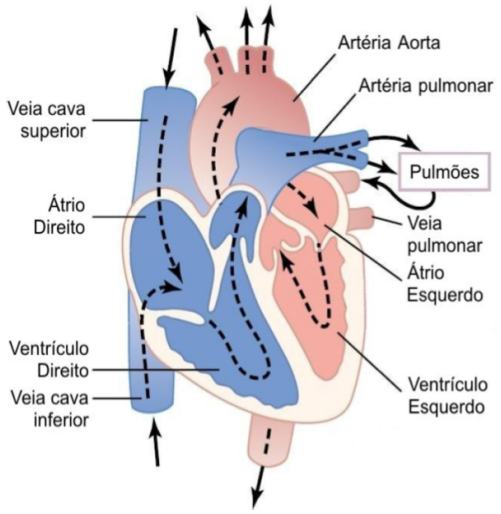

Figura 3: Fluxo sanguíneo através das câmaras e válvulas cardíacas - adaptado de [27]

pulmões e a circulação periférica. Este sistema também permite que os ventrículos se contraiam simultaneamente, o que é essencial para que ocorra uma geração de pressão mais efetiva em todo o órgão. Caso ocorram disfunções nestes impulsos elétricos, arritmias e outras doenças de origem fatal podem ser ocasionadas, levando ao mal funcionamento do órgão e, consequentemente, de todo o organismo.

3.2 Fisiologia da membrana celular

As células do corpo humano possuem uma membrana que as envolve, a membrana celular (ou plasmática). Ela engloba a célula, definindo seus limites e separando o meio intracelular do extracelular. A membrana celular é a principal responsável pelo controle de entrada e saída de substâncias da célula assim como pela manutenção da concentração iônica intracelular, distinta da concentração do meio extracelular. A membrana celular, representada na Figura 4, separa estes meios e é composta por uma dupla camada lipídica com proteínas entre essas camadas.

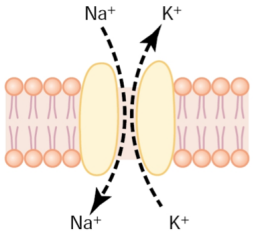

Figura 4: Estrutura de uma membrana celular - retirado de [4]

De acordo com [4], em alguns pontos, chamados de canais protéicos, as proteínas existentes nos meios intra e extracelular atravessam a dupla camada lipídica. Esse transporte é realizado através de comportas. São elas que regulam a entrada e saída de íons na célula. Os canais protéicos possuem permeabilidade seletiva. No caso dos canais de sódio, que apresentam cargas negativas em sua parede interna, as comportas se localizam na face da membrana voltada para o meio extracelular, e os íons de carga positiva são atraídos para dentro da célula, enquanto que os de carga negativa são repelidos. Já nos canais de potássio as comportas estão localizadas na face da membrana voltada para o meio intracelular.

A diferença de composição química e elétrica nos fluidos intra e extracelular gera uma diferença de potencial na membrana, o potencial transmembrânico. Esta diferença de potencial tem papel importante na comunicação intercelular através de impulsos elétricos [38].

3.3 Potenciais de ação

A diferença de potencial na membrana celular gera fluxos iônicos que se propagam por toda a célula. A regulação do potencial transmembrânico através dos canais iônicos é uma das funções mais importantes da célula [31].

Potenciais elétricos estão presentes em quase todas as células do corpo humano [4]. Uma característica importante de algumas células, como as nervosas e as musculares, é o poder de transmitir sinais pelo tecido através desses impulsos elétricos. Assim, o funcionamento do sistema nervoso e da contração muscular, por exemplo, dependem da geração e propagação de sinais elétricos, isto é, do potencial de ação.

Segundo [38], para se entender melhor a transmissão dos sinais elétricos pelas células, faz-se a divisão de todos os tipos de células em dois grupos: as células excitáveis e as não-excitáveis. Determinadas células conseguem manter um potencial de equilíbrio estável. Para algumas delas, se é aplicada uma corrente elétrica em um curto período de tempo, o potencial transmembrânico é retornado diretamente ao equilíbrio depois que a corrente é removida. Essas células são chamadas não-excitáveis.

Em contrapartida, as células em que a injeção de uma corrente suficientemente forte faz com que o potencial transmembrânico percorra um caminho, denominado potencial de ação, antes de retornar ao repouso, são chamadas células excitáveis. Um exemplo de células que possuem essa caractéristica, são as células cardíacas, células secretoras,

células musculares e a maioria dos neurônios. Uma característica das células excitáveis é que elas podem responder completamente a um estímulo. Desta forma, um estímulo com amplitude suficiente pode ser distinguido de um simples ruído. Assim, o ruído é filtrado e o sinal é transmitido de maneira confiável [38].

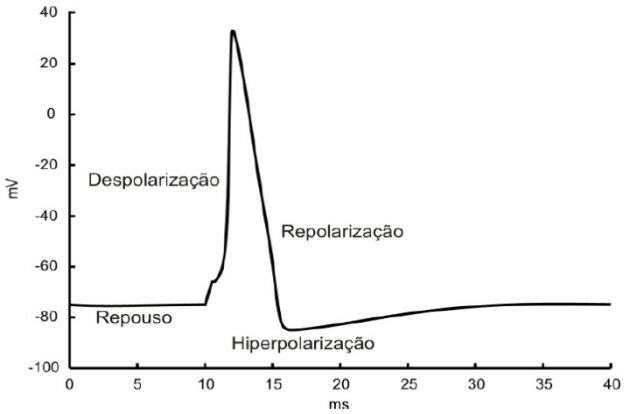

Figura 5: Potencial de ação - retirado de [3]

Baseado em [4], o potencial de ação (PA) é a variação repentina no potencial transmembrânico, causada pela passagem de corrente elétrica na célula. As fases do potencial de ação podem ser vistas na Figura 5 e são descritas a seguir: o potencial inicial, chamado de potencial repouso, gira em torno de -65mV. Ao se injetar uma determinada corrente na célula, observa-se uma brusca variação no potencial da membrana causada por um grande fluxo de íons Na^+ para dentro da célula, tornando esta célula muito permeável ao íon de Na^+. Esta fase em que o potencial cresce é chamado de despolarização, e não ultrapassa 1ms. Logo em seguida, os canais de sódio se fecham e os de potássio se abrem, gerando grande fluxo de íons K^+ para fora da célula, fazendo com que o potencial na membrana caia a valores próximos ao repouso, fase denominada de repolarização. A fase de hiperpolarização ocorre quando o potencial cai a valores menores que o potencial de repouso.

3.4 Modelo para a membrana celular

A principal característica da membrana celular é isolar o meio intra do extracelular. Por esse motivo, a membrana pode ser vista como um capacitor [28]. Sendo assim, o potencial na membrana V é proporcional à carga armazenada Q:

$$V = \frac{Q}{C_m}, \tag{3.1}$$

onde C_m é a capacitância da membrana.

De acordo com [38], o fluxo de íons muda a quantidade de carga armazenada pelo capacitor, assim como o potencial transmembrânico através da relação $Q = C_m V$ definida na Equação 3.1. Considerando a variação de carga no tempo, tem-se:

$$\frac{dQ}{dt} = C_m \frac{dV}{dt} \tag{3.2}$$

O termo $\frac{dQ}{dt}$ é a variação de carga no tempo, que corresponde a uma corrente, denominada de corrente capacitiva. A corrente total através da membrana celular é definida como a soma da corrente capacitiva com a corrente iônica que atravessa os canais,

$$I_m = I_{ion} + I_c \tag{3.3}$$

Substituindo a Equação 3.2 segue que

$$I_m = I_{ion} + C_m \frac{dV}{dt} \tag{3.4}$$

Se o circuito for fechado, não haverá transportes de íons na malha, então:

$$I_{ion} + C_m \frac{dV}{dt} = 0 \tag{3.5}$$

3.5 Potenciais de equilíbrio da membrana celular

A descrição da membrana celular como um circuito resistor-capacitor não considera a existência de uma diferença de potencial elétrico através da membrana celular [40]. Essa diferença de potencial é encontrada devido à diferente concentração de íons através da membrana e a diferença de permeabilidade da membrana a esses diversos íons. As equações de Nernst e Goldman-Hodgkin-Katz são usadas na modelagem da eletrofisiologia e descrevem essa diferença de potencial, complementando a descrição de [28].

O potencial resultado do equilíbrio entre a difusão iônica e o campo elétrico gerado é chamado de potencial de Nernst. Este potencial é determinado pela taxa de concentração do íon nos dois lados da membrana. A tendência do íon se difundir para uma direção é proporcional a esta taxa e consequentemente, quanto maior a taxa, maior o potencial de Nernst. A fórmula que define o potencial de Nernst é:

$$v = \frac{RT}{zF} ln \frac{C_i}{C_e}, \tag{3.6}$$

onde C_i e C_e representam a concentração iônica interna e externa, respectivamente, R é a

constante dos gases, T é a temperatura absoluta, z é a valência do íon e F é a constante de Faraday.

Para calcular o potencial quando a membrana é permeável a vários íons, deve-se considerar três fatores para cada íon, a polaridade da carga elétrica, a permeabilidade (P) da membrana e as concentrações (C) interna (i) e externa (e). A próxima fórmula, chamada de Goldman-Hodgkin-Katz, calcula o potencial de repouso dentro da membrana quando os íons Na^+, K^+ e Cl^- estão envolvidos [38]:

$$v\,(mV) = -\frac{RT}{zF} ln \frac{C_{Na_i^+} P_{Na^+} + C_{K_i^+} P_{K^+} + C_{Cl_e^-} P_{Cl^-}}{C_{Na_e^+} P_{Na^+} + C_{K_e^i} P_{K^+} + C_{Cl_i^-} P_{Cl^-}} \tag{3.7}$$

Esta fórmula reflete algumas propriedades, como a de que os íons de sódio, potássio e cloro são os mais importantes para o desenvolvimento do potencial de ação em tecidos musculares e nervosos. A concentração gradiente de cada um destes íons ajuda a determinar o potencial de membrana. Observa-se também que a permeabilidade dos canais iônicos sofrem rápidas mudanças durante um impulso elétrico.

3.6 Modelo de Hodgkin-Huxley

O modelo de Hodgkin e Huxley descreve a eletrofisiologia da membrana do axônio gigante de lula e foi desenvolvido a partir de medidas do comportamento elétrico passivo e ativo da célula [29]. A base da descrição do potencial de ação proposto por Hodgkin e Huxley é o comportamento dos canais de sódio e de potássio, e, a partir de dados experimentais, desenvolveram uma descrição matemática do comportamento da membrana. O modelo possui dois elementos básicos: a membrana celular e os canais iônicos que a permeiam, ilustrados na Figura 6.

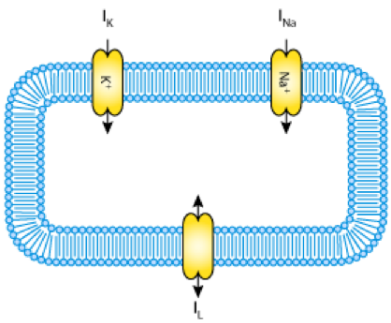

Figura 6: Fluxos de corrente pela membrana observados pelo modelo de Hodgkin Huxley - retirado de [5]

Segundo [4], a bi-camada lipídica é modelada eletricamente como um dielétrico de

capacitância C_m. Os canais iônicos são modelados como resistências não lineares dependentes da diferença de potencial $\phi = \phi_i - \phi_e$ sobre a membrana e da concentração de diversos íons, onde os íons de sódio e potássio formam as correntes iônicas mais importantes para a mudança de potencial da membrana. O circuito resultante desta combinação está na Figura 7.

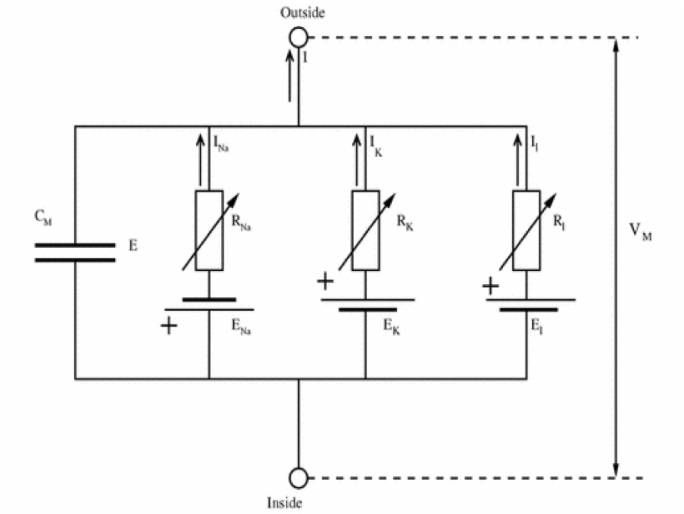

Figura 7: Circuito elétrico que descreve o fluxo de correntes - retirado de [39]

No modelo Hodgkin-Huxley três correntes são assumidas: $I_{ion} = I_{Na} + I_K + I_L$, sendo todas lineares em ϕ. Denotando a capacitância da membrana por C_m, obtém-se, a partir de 3.5 que:

$$-C_m \frac{d\phi}{dt} = I_{Na} + I_K + I_L \tag{3.8}$$

I_{Na} e I_K são, respectivamente, as correntes de sódio e potássio e I_L uma corrente de fuga constante. Todas elas são dadas por:

$$-C_m \frac{d\phi}{dt} = \sigma_{Na}(\phi - \phi_{Na}) + \sigma_K(\phi - \phi_K) + \sigma_L(\phi - \phi_L), \tag{3.9}$$

onde ϕ_{Na} e ϕ_K são os potenciais de Nernst para seus respectivos íons; σ_L é constante e, σ_{Na} e σ_K são as condutâncias dos canais, governadas por equações diferencias ordinárias.

3.7 Modelo de Bondarenko

O modelo de Bondarenko [2] representa miócitos ventriculares de camundongos e fornece informações quantitativas sobre o comportamento iônico dessas células. Ele descreve o comportamento elétrico de miócitos oriundos das regiões do ápice e septo cardíacos, mostrando também como a repolarização pode variar levando-se em conta as diferentes regiões

do coração. O modelo de Bondarenko é derivado do modelo de Hodgking-Huxley, porém, diferencia-se deste por utilizar cadeias de Markov [33] para representar a estrutura dos canais iônicos. Basea-se em um conjunto de 41 equações diferenciais ordinárias ($EDOs$) que simulam correntes iônicas, bombas e a homeostase celular, de modo a reproduzir um potencial de ação.

Este trabalho utiliza o modelo de Bondarenko, modelo este que também tem sido utilizado na modelagem computacional da doença de Chagas [7].

4 O problema de ajuste automático

Neste capítulo é abordado o problema inverso associado ao ajuste automático dos modelos celulares utilizados na modelagem da eletrofisiologia cardíaca.

Como observado no capítulo 3, os modelos celulares existentes reproduzem médias de características relevantes à atividade elétrica observada em determinados experimentos. Portanto, se existir um conjunto restrito de dados experimentais, que foram coletados a partir de experimentos com células cardíacas e um modelo matemático que faz a simulação do comportamento eletrofisiológico deste tipo de célula, faz-se necessário a utilização de ferramentas apropriadas para ajustar o modelo celular, de modo que este modelo consiga reproduzir os dados experimentais previamente coletados [38].

Este trabalho faz uso de uma metodologia baseada na resolução do seguinte problema inverso: Seja um potencial de ação ou um conjunto de potenciais de ação conhecidos: PA_c. Deseja-se ajustar as condutividades máximas v de um dado modelo celular, para que este seja capaz de reproduzir características de PA_c.

Portanto, o problema consiste em encontrar um vetor de condutividades iônicas v, tal que:

$$min_v F(v, PA_c) \tag{4.1}$$

A função F_1, representada a seguir, foi formulada inicialmente por [41] considerando apenas um potencial de ação PA_c disponível:

$$F_1(v, PA_c) = \sum_{i=1}^{n} |PA_c(i) - PA_s(v, i)| \tag{4.2}$$

onde v é um vetor de condutividades, n é o número de discretizações do PA, $PA_c(i)$ e $PA_s(v, i)$ são os valores no instante $i * dt$ do PA conhecido e do PA simulado pelo vetor de condutividades v, respectivamente; dt é a taxa de amostragem dos potenciais de ação.

Em experimentos reais, é comum se deparar com diversas variantes das características dos PAs. Como a Equação 4.2 leva em consideração apenas um PA_c conhecido, a

mesma acaba se tornando inviável para este tipo de problema que possui dados de vários potenciais de ação conhecidos. Visto isso, [38] formulou uma descrição mais completa, descrita na Equação 4.3. A nova função $F_2(v, PA_c)$ agora leva em consideração várias características importantes do PA. Característicais tais como o potencial de repouso, o pico do potencial transmembrânico, a derivada máxima ($\frac{dv}{dt}$), a amplitude (diferença entre o pico e o potencial de repouso), a duração do potencial de ação dada como a diferença entre o tempo de ativação (instante da derivada máxima) e o tempo o valor mais negativo do potencial após a fase de repolarização que a sucede; DPA30, 50 e 90, são as durações do potencial de ação a 30%, 50% e 90% da fase de repolarização, calculado usando a diferença entre o tempo de ativação e o tempo de repolarização. Estas fases do potencial de ação foram descritas na sessão 3.3.

$$\begin{aligned} F_2(v, PA_c) = & \frac{1}{DPA30_x(PA_c)}|DPA30_m(PA_c) - DPA30(v)| + \\ & \frac{1}{DPA50_x(PA_c)}|DPA50_m(PA_c) - DPA50(v)| + \\ & \frac{1}{DPA90_x(PA_c)}|DPA90_m(PA_c) - DPA90(v)| + \\ & \frac{1}{Amplitude_x(PA_c)}|Amplitude_m(PA_c) - Amplitude(v)| + \\ & \frac{1}{dmax_x(PA_c)}|dmax_m(DPA_c) - dmax(v)| + \\ & \frac{1}{repouso_x(PA_c)}|repouso_m(PA_c) - repouso(v)| \end{aligned} \tag{4.3}$$

onde v é um vetor de condutividades, $*_m$ é a média das características em questão extraídas de PA_c, $*(v)$ é o valor da característica do PA simulado usando o vetor de condutividades v e $*_x$ é descrita pela Equação 4.4; * pode ser $DPA30$, $DPA50$, $DPA90$, *Amplitude*, *dmax* ou *repouso*. Para o caso de múltiplos PA_c, $*_x$ é o desvio padrão das medidas da característica *. Para o caso de uma única medida PA_c, $*_x$ é o próprio valor da medida da característica *. Ou seja,

$$*_x = \begin{cases} dp(*) & \text{se} \quad dp(*) > 0 \\ * & \text{se} \quad dp(*) = 0 \end{cases} \tag{4.4}$$

onde $dp(*)$ é o desvio padrão das medidas da característica * de PA_c.

Para a resolução do problema inverso descrito pela Equação 4.1, este trabalho utiliza uma metaheurística baseada em evolução conhecida como algoritmo genético.

4.1 Algoritmos genéticos

Os Algoritmos Genéticos são uma família de modelos computacionais inspirados na evolução, que incorporam uma solução potencial para um problema específico, numa estrutura semelhante à de um cromossomo, aplicando operadores de seleção, cruzamento (*crossover*) e mutação a essas estruturas de forma a preservar informações críticas relativas à solução do problema.

Os AG's foram criados por John Holland nos anos 60, e seu objetivo inicial era estudar formalmente os fenômenos de adaptação que ocorrem na natureza e desenvolver maneiras para que os mecanismos de adaptação natural fossem importados para os sistemas de computação [12]. Segundo Holland, os algoritmos genéticos são programas de computador que evoluem de uma forma similar à seleção natural e podem resolver problemas complexos que até mesmo seus criadores não entendem completamente [30].

Normalmente os AG's são vistos como otimizadores de funções, embora a quantidade de problemas para os quais se aplicam seja bastante abrangente. Geralmente indicados para solução de problemas de otimização complexos, uma das vantagens de um algoritmo genético é a simplificação que eles permitem na formulação e solução deste tipo de problema. AG's simples normalmente trabalham com descrições de entrada formadas por cadeias de *bits* de tamanho fixo. Outros tipos de AG's, utilizados em Programação Genética, podem trabalhar com cadeias de *bits* de tamanho variável. Os algoritmos genéticos possuem um paralelismo implícito, decorrente da avaliação independente de cada uma dessas cadeias de *bits*, ou seja, pode-se avaliar independentemente a viabilidade de um conjunto de parâmetros para a solução do problema de otimização em questão.

O AG de Holland é um método para evoluir uma população de indivíduos para uma nova população usando um tipo de "seleção natural" juntamente com operadores inspirados na genética como cruzamento (*crossover*) e mutação. Cada indivíduo é representado por um cromossomo que consiste de "genes" (ex. *bits*) e cada gene é uma instância de um "alelo" particular (ex. 1 ou 0). Esses indivíduos são, então, avaliados por uma função objetivo, que tem por finalidade calcular a aptidão bruta de cada indivíduo e fornecer um valor a ser usado para o cálculo de sua probabilidade de ser selecionado para reprodução. Maiores probabilidades estão relacionadas aos cromossomos que representam uma melhor solução para o problema de otimização. Estes são chamados de mais aptos. Em geral, os cromossomos mais aptos geram mais descendentes que os menos aptos. O cruzamento promove a troca de subpartes dos cromossomos, na tentativa de imitar a recombinação

biológica entre dois organismos. Este processo gera um novo cromossomo, que possui características provenientes de cada um de seus pais. A mutação pode ser aplicada sobre um cromossomo para modificar, de forma aleatória, alguns de seus "alelos" [32].

Novas gerações de possíveis soluções do problema vão sendo geradas e o processo descrito acima continua se repetindo até que uma solução satisfatória seja encontrada. Na Figura 8 é apresentado um ciclo de um algoritmo genérico.

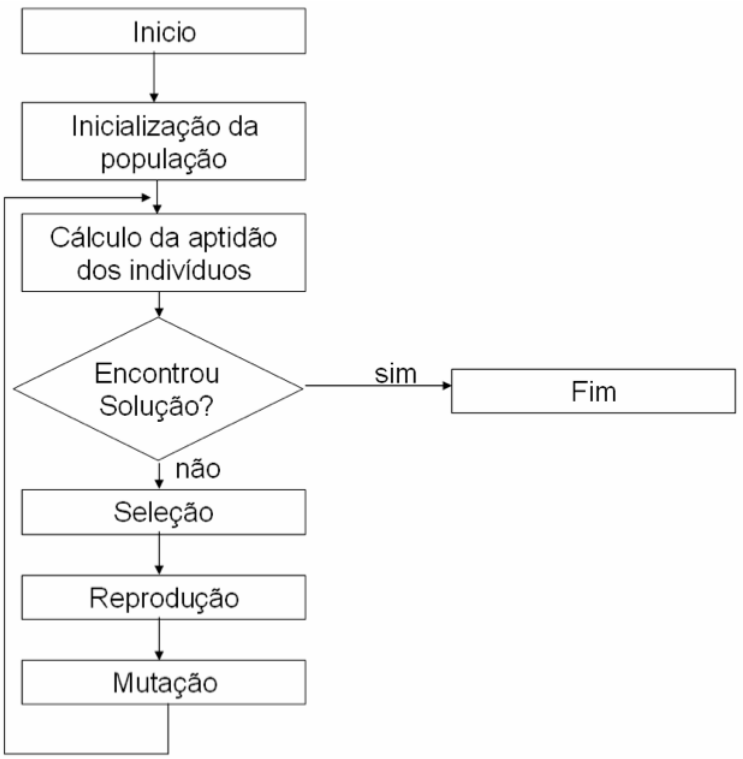

Figura 8: Pseudo-código de um ciclo do Algoritmo Genético - retirado de [32]

4.2 Metodologia

A metodologia utilizada neste trabalho é apresentada a seguir. [38] implementou um algoritmo genético para o ajuste automático das condutividades iônicas. Cada indivíduo é representado por um vetor de números reais, que são os valores das condutividades. Estas serão utilizadas posteriormente como parâmetro para a geração de um PA através de um modelo celular escolhido. A população inicial é gerada de forma aleatória. O critério de seleção dos indivíduos é por posição (*rank*). Uma lista ordenada informa quais indivíduos serão escolhidos para reprodução, onde os primeiros são aqueles que possuem maior aptidão. O esquema de reprodução é geracional, desta forma, todos os indivíduos da população serão substituídos por seus descendentes. Isso pode ocasionar a perda de bons

indivíduos. Visando este problema, o esquema de elitismo também foi implementado, mantendo o indivíduo mais apto na próxima geração. O método de *crossover* utilizado é o BLX-α e a mutação é não-uniforme.

O modelo computacional de [2] foi adotado neste trabalho. O mesmo foi representado computacionalmente utilizando a linguagem CellML [5]. Para a resolução das equações diferenciais ordinárias, a ferramenta AGOS (*Application Program Interface Generator for ODE Solution*) [1] foi utilizada. A ferramenta AGOS constrói automaticamente uma classe C++ orientada a objeto, permitindo a configuração de condições iniciais e de outros parâmetros, além da resolução numérica de problemas de valor inicial descritos em linguagens CellML ou MathML. O AGOS está disponível no portal FISIOCOMP [1] [13].

O paralelismo adotado é o mestre-escravo. Neste esquema, os escravos são responsáveis pelo cálculo da aptidão dos indivíduos, tarefa esta computacionalmente cara. Em um primeiro momento, a divisão do trabalho foi desenvolvida em [38] e é chamada síncrona. Nessa abordagem, o mestre faz a divisão igualitária do número de indivíduos entre os escravos, obrigando que o número de indivíduos deva ser múltiplo do número de escravos utilizados na computação. Deste modo, o mestre deve esperar todos os escravos calcularem a aptidão dos indivíduos e enviarem seus resultados ao mestre, para somente depois ser iniciada uma nova geração. Em um *cluster*, onde geralmente as máquinas possuem configurações semelhantes, essa estratégia era suficiente, pois o tempo de cálculo gasto por um escravo seria praticamente o mesmo tempo gasto nos demais. Mas como a grade é um ambiente altamente heterogêneo, ou seja, existem máquinas de diversas configurações, as máquinas mais rápidas teriam de esperar as mais lentas terminarem seu trabalho para somente depois receberem mais trabalho.

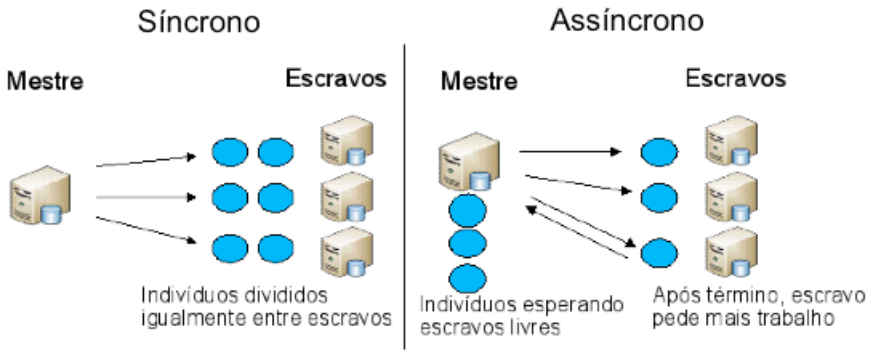

Figura 9: Comparação de divisão do trabalho mestre-escravo (síncrono x assíncrono) - adaptado de [4]

[1] http://www.fisiocomp.ufjf.br/

Na tentativa de evitar o desperdício desses recursos computacionais, [4] desenvolveu o modo de divisão de trabalho assíncrono. A Figura 9 ilustra os dois esquemas de divisão de trabalho. Nesta abordagem, o mestre envia somente um indivíduo por vez para cada escravo, e assim que este termina seu cálculo, o mestre o envia um novo indivíduo. Desta forma, escravos com maior poder de processamento receberão mais indivíduos que escravos com menor poder de processamento. Isso contribui para a diminuição do número de máquinas em estado *idle*, ou seja, em estado de espera, decorrente da diferença de tempo gasto no cálculo dos indivíduos por cada escravo.

A aplicação alvo para o porte utiliza a abordagem assíncrona para a divisão de trabalho, visto que este esquema faz um melhor uso dos recursos disponibilizados por ambientes altamente heterogêneos. Tal aplicação já se encontrava devidamente paralelizada, permitindo seu porte para um ambiente de grade. Para executar tal trabalho, foram realizados estudos sobre grades computacionais e o *middleware* gLite, responsável pela gerência do ambiente EELA-2. A linguagem JDL que descreve as necessidades de recursos da tarefa na grade também foi estudada a fundo, fornecendo uma base sólida para a criação do arquivo utilizado neste trabalho de porte.

Para submeter a tarefa na grade, faz-se necessária a criação de um arquivo JDL. Este é ilustrado na Figura 10.

```
1  [
2      JobType        = "MPICH";
3      CPUNumber      = 65;
4      Executable     = "mpi-start-wrapper.sh";
5      Arguments      = "ag mpich";
6      StdOutput      = "ag.out";
7      StdError       = "ag.err";
8      InputSandbox   = {"mpi-start-wrapper.sh" , "mpi-hooks.sh"};
9      OutputSandbox  = {"ag.err" , "ag.out"};
10     Requirements   = (((other.GlueCEInfoLRMSType == "PBS") ||
                          (other.GlueCEInfoLRMSType == "LSF")) &&
11                        (other.GlueCEUniqueID == "ce-eela.ciemat.es:
12                         2119/jobmanager-lcgpbs-prod_eela"));
13 ]
```

Figura 10: Arquivo JDL utilizado no trabalho

O valor do rótulo *JobType* é MPICH, já que a aplicação utiliza o padrão MPI para comunicação entre o mestre e os escravos. O executável neste caso, não é a própria aplicação mas sim um *script* que realiza uma série de preparativos para que a computação possa ser realizada de forma correta. Ele será descrito adiante. Os argumentos utilizados são a própria aplicação e o padrão de comunicação MPICH. Em *Requirements* algumas

restrições são impostas ao trabalho, como por exemplo, a definição de qual CE será utilizado para execução da tarefa.

```
1  #!/bin/bash
2  export LFC_HOST=lfc.eela.ufrj.br
3  export LFC_HOME=/grid/prod.vo.eu-eela.eu/peters/ag/assincrono
4  export LCG_GFAL_INFOSYS=bdii.eela.ufrj.br:2170
5  export LCG_CATALOG_TYPE=lfc
6  MY_EXECUTABLE=ag
7  MPI_FLAVOR=MPICH
8  MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR | tr '[:upper:]' '[:lower:]'`
9  eval MPI_PATH=`printenv MPI_${MPI_FLAVOR}_PATH`
10 eval I2G_${MPI_FLAVOR}_PREFIX=$MPI_PATH
11 export I2G_${MPI_FLAVOR}_PREFIX
12 touch $MY_EXECUTABLE
13 export I2G_MPI_APPLICATION=ag
14 export I2G_MPI_APPLICATION_ARGS="50 64 dadosControle30.txt"
15 export I2G_MPI_TYPE=mpich
16 export I2G_MPI_PRE_RUN_HOOK=mpi-hooks.sh
17 export I2G_MPI_POST_RUN_HOOK=mpi-hooks.sh
18 $I2G_MPI_START
```

Figura 11: *Script* com os ajustes para a execução do simulador

Além da criação de um arquivo contendo a descrição da tarefa, foram também necessários alguns ajustes para que a aplicação pudesse executar corretamente na grade. Um *script* baseado em [34], representado pela Figura 11, foi utilizado para fazer toda a configuração do ambiente antes que a execução pudesse ser iniciada. Neste arquivo, algumas variáveis de ambiente são corretamente definidas, como $LCG\_GFAL\_INFOSYS$ e LFC_HOST. Logo após outras variáveis de ambiente que definem caminhos de diretórios e arquivos foram ajustados.
Os parâmetros de entrada utilizados pela aplicação são especificados na linha 14:

- export I2G_MPI_APPLICATION_ARGS=”50 64 dadosControle30.txt”,

onde 50 é o número de gerações de indivíduos; 64 define a quantidade de indivíduos da população e dadosControle30.txt é o arquivo de controle com informações sobre a curva do potencial de ação utilizado. Tais informações compreendem o potencial de repouso, o pico do potencial transmembrânico entre outras características do PA vistas no capítulo 4. A implementação do MPI utilizada é especificada em $I2G\_MPI\_TYPE$, e as funções que serão executadas antes e depois da aplicação são descritas nas variáveis de ambiente, $I2G\_MPI\_PRE\_RUN\_HOOK$ e $I2G\_MPI\_POST\_RUN\_HOOK$ respectivamente. Essas funções são ilustradas na Figura 12.

Finalmente, na linha 18, a computação é iniciada com $\$I2G\_MPI\_START$.

Um outro *script* foi utilizado para definir as ações necessárias antes e após o término da execução da aplicação, como tratamento dos arquivos por exemplo. Este *script* possui duas funções especiais chamadas *pre_run_hook* e *post_run_hook*. A função *pre_run_hook* é responsável por ajustar corretamente as variáveis de ambiente e fazer a cópia dos arquivos necessários para a execução do *SE* para a máquina local que está chamado a função. Após copiá-los, a função também altera a permissão dos arquivos para execução, de modo que estes possam ser executados pelas máquinas participantes da computação. A função *post_run_hook* somente imprime a mensagem "Execução terminada." e também faz a listagem do diretório, onde podem ser identificados os arquivos que ali estão presentes. Esse *script* pode ser visto na Figura 12.

```
1  pre_run_hook(){
2      export LCG_GFAL_INFOSYS=bdii.eela.ufrj.br:2170
3      export LCG_CATALOG_TYPE=lfc
4      export LFC_HOST=lfc.eela.ufrj.br
5      export LFC_HOME=/grid/prod.vo.eu-eela.eu/peters/ag/assincrono
6
7      LFNStoredFile1="dadosControle30.txt"
8      LFNStoredFile2="condicoes_iniciais"
9      LFNStoredFile3="ag"
10
11     lcg-cp --vo prod.vo.eu-eela.eu lfn:$LFNStoredFile3 ag
12     lcg-cp --vo prod.vo.eu-eela.eu lfn:$LFNStoredFile1 dadosControle30.txt
13     lcg-cp --vo prod.vo.eu-eela.eu lfn:$LFNStoredFile2 condicoes_iniciais
14
15     chmod 777 ag
16     chmod 777 dadosControle30.txt
17     chmod 777 condicoes_iniciais
18
19     return 0
20 }
21
22 post_run_hook(){
23     echo "Execucao terminada."
24     ls -l
25     return 0
26 }
```

Figura 12: *Script* com funções que são executadas antes e depois da aplicação

5 Resultados

O apêndice A foi utilizado com o objetivo de realizadar a submissão dos *jobs* do simulador da eletrofisiologia cardíaca na grade do EELA-2. A aplicação foi executada com diversas configurações de números de nós para que pudesse ser avaliado seu desempenho. Todos os testes realizados utilizaram 50 gerações e uma população de 64 indivíduos para os parâmetros da aplicação.

Como já mencionado, a aplicação faz uso da biblioteca MPICH para comunicação entre processos. Entretanto, alguns problemas foram encontrados devido à falta de grande número de máquinas com tal biblioteca devidamente instalada e configurada. Ao submeter um *job* para determinados *sites* (CEs) que, teoricamente possuíam os requisitos especificados na tarefa, como por exemplo, suporte ao padrão MPICH, o *job* não conseguia ser executado pelo fato das máquinas do *site* não possuírem o padrão citado devidamente instalado. Levou-se bastante tempo para encontrar CEs que dispunham dos requisitos especificados na tarefa e possibilitavam o perfeito processamento da aplicação. Foram realizados vários testes em diversos *sites* de modo a selecionar aqueles que poderiam suportar efetivamente a execução aplicação.

No início dos experimentos, quando o número de máquinas para a execução estava configurado com 1, 2 e 4 nós, o *middleware* gLite cancelava a tarefa que já estava há algumas horas em processamento por esta ultrapassar o tempo limite de execução estipulado, já que para os parâmetros utilizados e número de nós especificados, a aplicação demanda várias horas de computação. Dada essa restrição, foram utilizadas configurações de 8, 16, 32 e 64 nós. Para cada instância de configuração, a aplicação foi executada cinco vezes. Isso foi feito para avaliar o tempo médio gasto na execução para cada número de nós e fazer o cálculo do desvio padrão dos tempos recolhidos, de forma a assegurar a confiabilidade dos resultados obtidos. Ao final de cada execução, os tempos de execução gastos foram somados e divididos por cinco, obtendo a média aritmética. A Figura 13 ilustra a média dos tempos coletados.

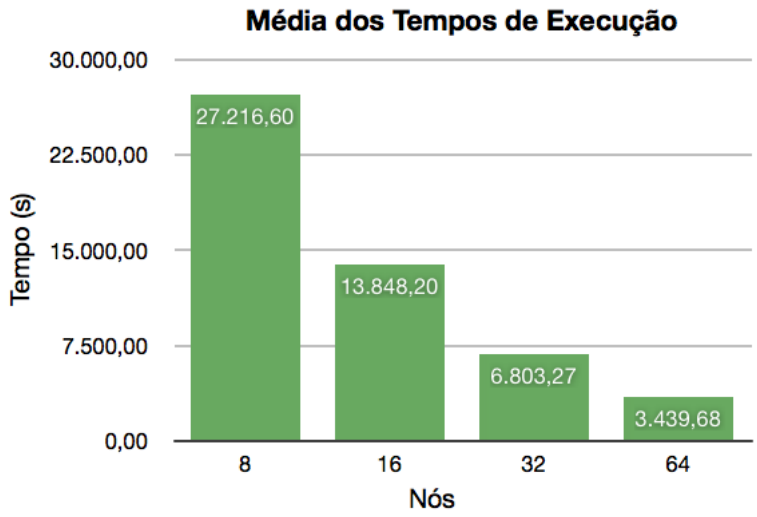

Figura 13: Gráfico da Média dos tempos de execução coletados

Também foi realizado o cálculo da aceleração (*speedup*). A aceleração informa o quão mais rápida uma aplicação será executada à medida que mais processadores são incorporados à sua execução. Para o cálculo da aceleração S_n, utiliza-se a Equação 5.1, que é dada pela divisão do tempo gasto na execução sequencial pelo tempo da execução em n nós:

$$S_n = \frac{T_1}{T_n}, \tag{5.1}$$

onde T_n é a média de tempo gasto na execução utilizando n nós e T_1 é o tempo médio gasto na execução sequencial.

Como o mínimo de nós utilizados que possibilitaram a plena execução do simulador na grade foi de um total de 8 e não de 1, o cálculo da aceleração 5.1 teve de ser adaptado para levar em conta não o tempo de execução sequencial, mas sim o tempo de execução em 8 nós. Esse detalhe impossibilitou uma melhor análise de desempenho do simulador. O ideal seria realizar a comparação de tempos a partir do momento que a aplicação deixa de ser executada sequencialmente e passa para a execução com um número maior de máquinas. Seria interessante verificar qual seria seu tempo gasto na execução sequencial e compará-lo com o tempo gasto, por exemplo, em dois nós de computação, de modo a analisar o ganho de tempo de uma configuração em relação à outra.

Após realizar todos os experimentos, os tempos de execução para cada número de nós foram coletados.

A Tabela 1 foi montada com todas as métricas anteriormente citadas para os dados

obtidos com a implementação assíncrona do simulador, fornecendo estatísticas para cada número de nós utilizados na aplicação quando esta é executada no ambiente de grade computacional.

Tabela 1: Tabela com informações sobre as execuções do simulador

Número de nós	**Tempo médio (s)**	**Desvio padrão**	**Aceleração**
8	27.216,60	42,85 (0,15%)	-
16	13.848,20	43,86 (0,31%)	1,96
32	6.803,27	29,48 (0,43%)	4,00
64	3.439,68	21,82 (0,63%)	7,91

À medida que é aumentado o número de nós, pode-se notar que o tempo cai na mesma proporção. Isso pode ser conferido na tabela acima, onde a execução da aplicação com 8 máquinas por exemplo, leva cerca de 27.216 segundos, e ao dobrar o número de nós para 16, a execução leva cerca de apenas 13.848 segundos para ser finalizada, quase que a metade do tempo gasto na configuração anterior. Os desvios padrões encontrados são baixos, não havendo grande discrepância nos valores dos tempos coletados experimentalmente.

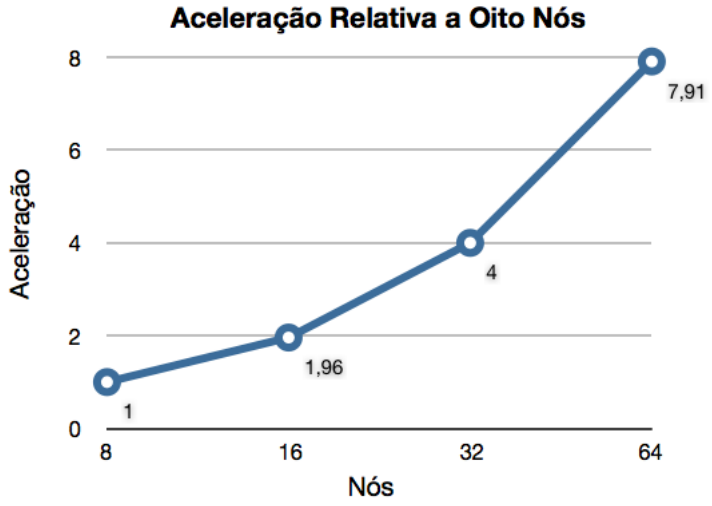

Figura 14: Gráfico de Aceleração relativa a oito nós

Baseado na Tabela 1, a Figura 14 ilustra o gráfico da aceleração, onde os pontos marcados representam a aceleração obtida em relação à configuração de oito nós, para cada número de nós utilizados distribuídos ao longo do eixo-x. Pode-se concluir que as acelerações obtidas são praticamente lineares, o que indica que à medida que dobramos o número de nós, o tempo necessário para a computação cai praticamente na mesma proporção.

6 Conclusão

Este trabalho do porte de um simulador da eletrofisiologia cardíaca para um ambiente de grade computacional foi realizado com algumas imposições propostas pelo ambiente utilizado. Apesar do porte estar em um estágio inicial, os resultados são promissores, indicando uma aceleração praticamente linear com relação ao número de processadores utilizados na computação.

Infelizmente, ao longo dos experimentos, algumas restrições impostas pelo ambiente EELA-2 aos seus usuários limitaram a realização de testes que seriam de grande interesse, como a utilização mais nós da grade, visto que pode-se localizar no ambiente de grade CEs com número de processadores da ordem de milhares.

A primeira restrição diz respeito exatamente à essa quantidade de nós que cada tarefa pode alocar. Este valor, pelo que pode-se observar, é uma fração dos nós disponíveis para computação em cada *site*. Se por um lado esta restrição evita que um usuário reserve para si todos os recursos disponíveis na grade, por outro lado cria limites para a quantidade de recursos computacionais que cada tarefa pode alocar.

Outro obstáculo encontrado é a existência de restrições no tempo total de uso das máquinas. O *middleware* gLite limita o tempo de computação disponível para a execução das tarefas. A execução com quatro nós, por exemplo, demora cerca de dezesseis horas, e apesar das tarefas terem sido submetidas para filas apropriadas (com altos tempos de execução) e dos certificados de *proxy* terem sido criados com duração de trinta e seis horas, a aplicação em questão era retirada do ambiente após cerca de doze horas de computação. Isso foi um empecilho para executar a aplicação com configurações com menos de oito nós, forçando a utilização de parâmetros diferentes dos originalmente utilizados em [38] como 500 gerações e com uma população de 128 indivíduos pelo menos. Devido às restrições impostas, o simulador teve de ser executado com o mínimo de oito e máximo de sessenta e quatro nós.

Durante o desenvolvimento do trabalho, tentou-se fazer o uso de um outro *middleware*

de controle da grade, o DIRAC. O projeto DIRAC é uma solução completa em grades computacionais para uma comunidades de usuários que desejem executar suas aplicações remotamente [8]. Uma característica que pode-se destacar deste *middleware* é a utilização de *Pilot Jobs* que são vistos como mediadores entre o *job* submetido e o CE onde ele será executado. De acordo com [9], tudo começa quando o usuário submete um *job*. Este então vai para uma fila e permanece nela até que seja selecionado por um *Pilot Job*. Este, por sua vez, somente faz a seleção de um *job* depois de certificar-se de que o ambiente de execução da tarefa esteja devidamente configurado e pronto para uso, garantindo assim que o *job* possa usufruir de todo o ambiente que se encontra em pleno funcionamento.

Infelizmente, por se encontrar ainda em fase de desenvolvimento, o DIRAC não funcionou corretamente como esperado, apresentando alguns problemas durante o processo de seleção realizado pelo *Job Pilot*, não conseguindo executar as tarefas submetidas à grade de tipo MPICH, padrão utilizado na aplicação.

Contudo, apesar das diversas limitações impostas, pode-se concluir que o ambiente de grade computacional é uma ferramenta de grande poder computacional que pode auxiliar na execução de aplicações que demandam alto desempenho. Como trabalhos futuros pretende-se avaliar o simulador fazendo o uso do *middleware* DIRAC e de outros ambientes de grade que não possuam tais limitações impostas pelo ambiente atual. Também seria de grande interesse portar outras aplicações do laboratório FISIOCOMP [13] para a grade.

Referências

[1] BARBOSA, C. B.; SANTOS, R. W.; AMORIM, R.; CIUFFO, L. N.; MANFROI, F.; OLIVEIRA, R. S.; CAMPOS, F. O. *A transformation tool for ode based models. Lecture Notes in Computer Science*, 2006.

[2] BONDARENKO, V. E.; SZIGETI, G. P.; BETT, G. C. L.; KIM, S. J.; RASMUSSON, R. L. *A computer model of the action potential of the mouse ventricular myocytes. American Journal of Physiology*, 2004.

[3] CAMPOS, F. O. *Modelagem computacional da eletrofisiologia cardíaca: O desenvolvimento de um novo modelo para células de camundongos e avaliação de novos esquemas numéricos*, 2008. Dissertação (Mestrado em Modelagem Computacional) - Mestrado em Modelagem Computacional, Universidade Federal de Juiz de Fora.

[4] CAMPOS, R. S. *A utilização de uma grade computacional para um problema inverso da eletrofisiologia cardíaca*, 2008. Monografia(Graduação) - Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora.

[5] CELLML, Disponível em: http://www.cellml.org, Acesso em: 05/10/2010.

[6] CIUFFO, L. N. *How to access EELA*, Disponível em: https://grid.ct.infn.it/twiki/bin/view/EELA2/HowToAccessEELA, Acesso em: 04/08/2010.

[7] COSTA, C. M. *Modelagem Computacional da Doença de Chagas*, 2008. Monografia(Graduação) - Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora.

[8] DIRAC, Disponível em: https://dirac.up.pt/DIRAC/, Acesso em: 03/10/2010.

[9] DIRAC, TSAREGORODTSEV, A.; BROOK, N.; RAMO, A. C.; CHARPENTIER, Ph.; CLOSIER, J.; COWAN, G.; DIAZ, R. G.; LANCIOTTI, E.; MATHE, Z.; NANDAKUMAR, R.; PATERSON, S.; ROMANOVSKY, V.; SANTINELLI, R.; SAPUNOV, M.; SMITH, A. C.; MIGUELEZ, M. S.; ZHELEZOV, A. *DIRAC3 - the new generation of the LHCb grid software*, 2009.

[10] EELA-2, Disponível em: http://www.eu-eela.eu/, Acesso em: 27/10/2010.

[11] EELA-2 Operations Centre, *Manipulate your certificate*, Disponível em: http://eoc.eu-eela.eu/doku.php?id=manipulate_your_certificate, Acesso em: 03/09/2010.

[12] EIBEN, A. E.; SMITH, J. E. *Introduction to Evolutionary Computing*. [S.l.]: Springer, 2003. 748 p.

[13] FISIOCOMP, Disponível em: http://www.fisiocomp.ufjf.br. Acesso em: 10/11/2010.

[14] FOSTER, I.; KESSELMAN, C.; TUECKE, S. *The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications*, 2001.

[15] FOSTER, I.; KESSELMAN C. *The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers*, 1998.

[16] GLITE, Disponível em: http://glite.web.cern.ch/glite/, Acesso em: 18/11/2010.

[17] GLITE, *Data Management*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 22/09/2010.

[18] GLITE, *Getting the Certificate*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 02/09/2010.

[19] GLITE, *LFC Commands*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 22/09/2010.

[20] GLITE, *PROXIES*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 14/09/2010.

[21] GLITE, *Single Job Submission*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 21/10/2010.

[22] GLITE, *The Job Description Language*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 14/10/2010.

[23] GLITE, *VOMS Proxies*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 24/10/2010.

[24] GLITE, *Workload Management*, Disponível em: https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html, Acesso em: 10/09/2010.

[25] GLOBUS, Disponível em: http://www.globus.org/, Acesso em: 18/11/2010.

[26] GRID CAFÉ, Disponível em: http://gridcafe.web.cern.ch, Acesso em: 27/10/2010.

[27] GUYTON, A. C.; HALL, J. E. *Textbook of medical physiology*. Philadelphia, Pennsylvania: Elsevier Inc., 2006.

[28] HILLE, B. *Ionic Channels of Excitable Membranes. Sinauer Associates*, 3 edição, 2001.

[29] HODGKIN, A. L.; HUXLEY, A. F. *A quantitative description of membrane current and its application to conduction and excitation. The Journal of Physiology*, 117: 500-557, 1952.

[30] HOLLAND, J. H. *Genetic algorithms. Scientific American*, 1992.

[31] KEENER, J.; SNEYD, J. *Mathematical Physiology*. Springer, 1998.

[32] MARTINS, D. M. S. *Algoritmos Genéticos Distribuídos Aplicados Ao Problema Inverso De Eletrofisiologia Cardíaca*, 2008. Dissertação (Mestrado em Ciências em Engenharia Civil), Universidade Federal do Rio De Janeiro.

[33] MARKOV, A. *Extension of the limit theorems of probability theory to a sum of variables connected in a chain*. In: HOWARD, R. (Ed.). *Dynamic Probabilistic Systems*. New York City: John Wiley and Sons, Inc., 1971. Volume I: Markov Models, cap. Appendix B, p. 552–577.

[34] MILLS, A. *MPI Jobs*, Disponível em: http://egee-uig.web.cern.ch/egee%2Duig/production_pages/MPIJobs.html, Acesso em: 06/11/2010.

[35] MINISTÉRIO DA SAÚDE. *Uma análise da mortalidade no Brasil e regiões*. 2004. Disponível em: http://portal.saude.gov.br/portal/saude/Gestor/visualizar_texto.cfm?idtxt=24421, Acesso em: 13/09/2010.

[36] OURGRID, Disponível em: http://www.ourgrid.org/, Acesso em: 18/11/2010.

[37] RAFFA, Robert B. et. al. *Farmacología Ilustrada*. Barcelona, Elsevier, 2008.

[38] OLIVEIRA, R. S. *Ajuste automático de modelos celulares apoiado por Algoritmos Genéticos*, 2008. Dissertação (Mestrado em Modelagem Computacional) - Mestrado em Modelagem Computacional, Universidade Federal de Juiz de Fora.

[39] ROCHA, B. M. *Um modelo de acoplamento eletromecânico do tecido cardíaco através do método dos elementos finitos*, 2008. Dissertação (Mestrado em Modelagem Computacional) - Mestrado em Modelagem Computacional, Universidade Federal de Juiz de Fora.

[40] SACHSE, F. B. *Computational Cardiology: Modeling of Anatomy, Electrophysiology and Mechanics*. Springer, 2004.

[41] SYED, Z. F.; VIGMOND, E. J.; NATTEL, S.; LEON, L. J. *Atrial cell action potential parameter fitting using genetic algorithms. Medical and Biological Engineering and Computing*, 43(5): 561-571, 2005b.

[42] UFF, *UFF Brazilian Grid Certification Authority*, Disponível em: http://brgridca.ic.uff.br, Acesso em: 16/11/2010.

APÊNDICE A – Roteiro de utilização da grade

Este roteiro tem o intuito de complementar o trabalho, de modo a fornecer informações de como acessar a grade e os comandos utilizados para o armazenamento de arquivos e submissão dos *jobs* no ambiente de grade computacional.

A.1 Acessando a grade

Para obter acesso à grade, deve-se possuir um certificado que é um identificador eletrônico [6]. Para obtê-lo, o usuário precisa requisitá-lo em uma IGTF (*International Grid Trust Federation*) associada a uma autoridade certificadora (CA - *Certification Authority*). Existe no Brasil, uma CA que está sediada na Universidade Federal Fluminense e os pedidos de certificados digitais podem ser realizados no site UFF *Brazilian Grid Certification Authority* [1].

Existem várias regras e passos para se conseguir um certificado digital. Tais regras são muito rigorosas, e consistem em enviar documentos de identidade e comprovante de matrícula ambos registrados em cartório. Também é necessário realizar uma vídeo-conferência com o responsável da CA. O próximo passo após obter o certificado, é se registrar em uma Organização Virtual (VO). A partir deste ponto é preciso criar uma conta para obter acesso à Interface do Usuário (UI), que é uma máquina *Unix* onde os usuários da grade possuem uma conta pessoal com o seu certificado instalado. É através desta máquina que acontece a autenticação, autorização e utilização dos recursos computacionais que a grade oferece. Para conseguir uma conta, o usuário deverá criar um par de chaves pública e privada. A chave privada será utilizada para autenticá-lo no acesso à UI sem a necessidade de uma senha, e a chave pública deverá ser enviada por *e-mail* ao responsável por gerenciar os usuários da grade [6]. O comando para criação de um par de chaves pública e privada em uma máquina *Unix* é dado abaixo:

[1] https://brgridca.ic.uff.br/

- ssh-keygen

Quando pedir uma *passphrase* não digite nada, apenas tecle *Enter* duas vezes para confirmar.

Após receber o *e-mail* de confirmação para utilização da grade, o usuário deverá exportar o seu certificado do *browser* para um local de armazenamento. O arquivo resultante normalmente possui a extensão ".p12". Neste ponto, se faz necessário converter o certificado de extensão ".p12" em arquivos de chaves pública e privada [11], para colocá-los no diretório ".globus" na interface de usuário da grade. O procedimento de conversão dos arquivos é facilmente realizado através dos comandos abaixo [18]:

- openssl pkcs12 -nocerts -in <certificado>.p12 -out userkey.pem
- openssl pkcs12 -clcerts -nokeys -in <certificado>.p12 -out usercert.pem

onde <certificado> é o nome do arquivo de extensão ".p12" exportado do *browser*.

Depois de ter copiado os arquivos de extensão ".pem" para o diretório ".globus", o certificado pode ser verificado utilizando o comando:

- grid-cert-info

O próximo passo é criar um certificado de *proxy* válido relacionado a uma VO. Sem o uso de um *proxy* não é possível fazer uso dos recursos da grade. Os comandos abaixo [23] servem para iniciar o *proxy* e verificar suas informações, respectivamente:

- voms-proxy-init - -voms prod.vo.eu-eela.eu
- voms-proxy-info

A.2 Manipulação de arquivos

Para gravar arquivos na grade, o usuário de uma determinada VO precisa ajustar adequadamente algumas variáveis de ambiente. Para este trabalho em especial, foram utilizadas os seguintes atributos:

- export LCG_GFAL_INFOSYS=bdii.eela.ufrj.br:2170

- export LCG_CATALOG_TYPE=lfc
- export LFC_HOST=lfc.eela.ufrj.br
- export LFC_HOME=/grid/prod.vo.eu-eela.eu/peters

A estutura de diretórios do LFC utiliza a seguinte convenção:

- /grid/<VO>/<caminhoArquivo>

onde usuários de um determinado VO terão acesso a leitura e gravação somente a partir do subdiretório <VO> correspondente. A partir deste ponto pode-se utilizar uma série de comandos para manipulação de pastas e arquivos.

Para a criação de um diretório, utiliza-se o comando abaixo:

- lfc-mkdir <localPasta>

onde <localPasta> é o caminho onde a pasta será criada respeitando a convenção citada acima, como por exemplo /grid/prod.vo.eu-eela.eu/novapasta.

Caso o usuário deseje verificar quais arquivos estão contidos no diretório <localPasta>, basta digitar:

- lfc-ls <localPasta>

Para a listagem dos *SEs* que armazenam os arquivos, utiliza-se o comando abaixo, onde <VO> é a organização virtual do usuário:

- lcg-infosites - -vo <VO> se

Para criar um arquivo em um *SE*, utiliza-se:

- lcg-cr - -vo <VO> -l lfn:<localArquivoDestino> -d <nomeSE> <localArquivoOrigem>

onde <localArquivoDestino> será o caminho remoto do arquivo, <localArquivoOrigem> é o local do arquivo de origem que será copiado para o *SE* <nomeSE>.

Para criar os arquivos utilizados neste trabalho e alterar suas permissões de uso, os comandos abaixo foram digitados:

- lcg-cr - -vo prod.vo.eu-eela.eu -l lfn:ag -d se-eela.ciemat.es ag
- lcg-cr - -vo prod.vo.eu-eela.eu -l lfn:condicoes_iniciais -d se-eela.ciemat.es condicoes_iniciais
- lcg-cr - -vo prod.vo.eu-eela.eu -l lfn:dadosControle30.txt -d se-eela.ciemat.es dadosControle30.txt
- lfc-chmod 777 ag
- lfc-chmod 777 condicoes_iniciais
- lfc-chmod 777 dadosControle30.txt

Todos os arquivos foram armazenados no *SE* se-eela.ciemat.es.

Mais comandos para manipulação de arquivos na grade podem ser conferidos em [17].

A.3 Subissão do trabalho

A última etapa deste trabalho foi a submissão dos *jobs* na grade. Para submeter um *job* de arquivo ".jdl" <job.jdl>, com uma delegação previamente criada <delegID>, utiliza-se o comando:

- glite-wms-job-submit -d <delegID> -o <arquivoID> <job.jdl>

Ao digitar o comando, um identificador (*jobID*) para o *job* é retornado e gravado no arquivo <arquivoID>. Esse identificador será utilizado para diversas ações, como visualizar o *status*, cancelar e obter o resultado do *job*, por exemplo. Depois de submetido, pode-se verificar qual o *status* do *job* digitando o comando a seguir, substituindo <jobID> pelo identificador obtido no comando anterior criado pela submissão da tarefa:

- glite-wms-job-status -i <jobID>

Os possíveis *status* são apresentados na Tabela 2:

Uma vez que o *status* seja **DONE**, o usuário pode obter as saídas do *job* de identificador <jobID> [22]:

- glite-wms-job-output -i <jobID>

Tabela 2: Tabela de possíveis *status* para um *job*

Status	**Descrição**
SUBMITTED	*job* submetido pelo usuário ainda não foi processado
WAITING	*job* esperando para ser aceito pelo WMS
READY	O *job* foi designado a um CE mas não foi transferido ainda
SCHEDULED	*job* esperando para ser processado na fila do CE
RUNNING	O *job* está sendo executado
DONE	*job* terminou a execução
ABORTED	*job* abortado pelo WMS por algum motivo em específico
CANCELED	*job* cancelado pelo usuário
CLEARED	O *Output Sandbox* foi transferido para a UI

Caso o usuário deseje cancelar uma tarefa de identificador <jobID>, deve-se utilizar:

- glite-wms-job-cancel -i <jobID>